Brazilian Journal of Development

Reduzindo Erros Comuns em Modelos de Requisitos Baseados na Linguagem i*: Uma Abordagem Ontológica

Mitigating Common Errors in istar-based Requirement Models: An Ontological Approach

DOI:10.34117/bjdv6n2-265

Recebimento dos originais: 30/12/2019
Aceitação para publicação: 26/02/2020

Renato de Freitas Bulcão-Neto
Doutor em Ciência da Computação, pelo ICMC-USP
Instituição: Instituto de Informática, Universidade Federal de Goiás
Endereço: Alameda Palmeiras, Quadra D, Câmpus Samambaia, 74690-900, Goiânia-GO, Brasil
E-mail: rbulcao@ufg.br

Heyde Francielle do Carmo França
Mestre em Ciência da Computação, pelo INF-UFG
Instituição: Instituto Federal Goiano, Campus Rio Verde
Endereço: Rodovia Sul Goiana, Km 01, Zona Rural, 75901-970, Rio Verde-GO, Brasil
E-mail: hheyde@gmail.com

RESUMO
A linguagem de modelagem i* representa os objetivos do sistema e da organização e traz diversas vantagens. Entretanto, o uso indevido de construtores da i*, ambiguidades na interpretação desses construtores e a complexidade dos modelos i* têm sido frequentemente relatados. Este artigo descreve uma abordagem baseada em ontologias que reduz em aproximadamente 70% esses problemas na construção de modelos de requisitos i*.

Palavras-chave: Engenharia de Requisitos, Linguagem i*, Ontologia, Ferramenta, Avaliação.

ABSTRACT
The istar modeling language represents the system's and the organization's goals and brings several advantages. However, the misuse of istar constructs, ambiguities on the interpretation of those constructs, and the complexity of the resulting istar models are frequent. This work presents an ontology-based approach that demonstrates an approximate coverage of 70% of the most common errors while constructing istar-based requirement models.

Keywords: Requirements Engineering, Istar language, Ontology, Tool, Evaluation.

1 INTRODUÇÃO

A linguagem i* tem demonstrado uma série de benefícios para a modelagem de requisitos de software (HORKOFF; YU, 2011), tais como representar o entendimento das intenções e motivações dos membros da organização envolvida, promover rastreabilidade entre as funcionalidades do sistema e os requisitos e permitir a análise de caminhos alternativos que podem ser seguidos para a satisfação de requisitos (GUIZZARDI; FRANCH; GUIZZARDI, 2012).

Brazilian Journal of Development

Apesar dessas vantagens, alguns trabalhos (SOUZA *et al.*, 2013; MALTA *et al.*, 2011) apontam problemas quanto à qualidade dos modelos produzidos na linguagem i*, como a complexidade dos modelos resultantes, erros típicos de mau uso de construtores e ambiguidades na interpretação desses construtores. Estes dois últimos problemas estão inter-relacionados, pois o mau uso dos construtores, em geral, acontece em função de erros de interpretação tanto da sintaxe quanto da semântica da linguagem i* (GUIZZARDI; FRANCH; GUIZZARDI, 2012).

Há trabalhos que exploram a sintaxe e a semântica da linguagem i*, como a linguagem *iStarML* (CARES *et al.*, 2011) e a ontologia *OntoiStar+* (NAJERA *et al.*, 2013a). Porém, essas soluções não resolvem erros típicos em modelos i* de requisitos, como os erros na definição de atores, dependências, relacionamentos entre atores, etc. Esses erros comumente encontrados estão resumidos em um catálogo de erros frequentes da linguagem i* (SOUZA *et al.*, 2013).

Este artigo visa apresentar uma solução baseada em ontologias para o tratamento de erros comuns em modelos de requisitos construídos na linguagem i*. Para isso, a ontologia *OntoiStar+* (NAJERA *et al.*, 2013a) foi estendida, a qual denomina-se *OntoiStar-NG*, com a escrita de axiomas e restrições seguindo a semântica da linguagem OWL e o estudo dos construtores da linguagem i*. Após a criação da ontologia OntoiStar-NG, esta foi validada por meio da modelagem de conceitos do clássico cenário *Media Shop* (CASTRO; KOLP; MYLOPOULOS, 2002) para verificar se é possível criar um modelo ontológico de requisitos com os erros catalogados (SOUZA *et al.*, 2013).

Visando dar o apoio ferramental na construção de modelos i* baseados na *OntoiStar-NG*, foi estendida a ferramenta *TAGOOn+* (*Tool for the Automatic Generation of Organizational Ontologies*) desenvolvida por Najera *et al.* (2013b), a qual denomina-se *TAGOOn-NG*. O objetivo com o uso da ferramenta *TAGOOn-NG* é possibilitar a transformação automática de modelos de requisitos representados em *iStarML* para uma instância da *OntoiStar-NG*, porém com o mínimo dos erros encontrados em modelos i* descritos em *iStarML*. Para validação da ferramenta *TAGOOn-NG* foi utilizado um cenário de Universidade (NAJERA, 2011).

Como resultado do uso conjunto da ontologia *OntoiStar-NG* e da ferramenta *TAGOOn-NG*, conseguiu-se tratar 11 dos 15 erros descritos no catálogo, contribuindo com uma melhora na qualidade dos modelos de requisitos escritos em i*.

2 MATERIAIS E MÉTODOS

Esta seção apresenta materiais para a realização deste trabalho: a linguagem i*, o catálogo de erros comuns, a ontologia *OntoiStar+* e a ferramenta *TAGOOn+*. Em seguida, são descritos os métodos usados para desenvolver a ontologia *OntoiStar-NG* e a ferramenta *TAGOOn-NG*.

Brazilian Journal of Development

2.1 LINGUAGEM i*

A linguagem i* caracteriza-se por modelar os objetivos e as intenções dos atores e suas dependências dentro de um contexto organizacional, permitindo descrever os relacionamentos estratégicos e intencionais em alto nível de detalhamento. Isso auxilia a compreender melhor qual o âmbito do problema, assim como a identificar mais facilmente quais as preocupações dos usuários e quais os requisitos que estes desejam ver realizados no sistema (HORKOFF *et al.*, 2008).

A linguagem i* inclui dois tipos de modelo: o SD (*Strategic Dependency*) e o SR (*Strategic Rationale*). O modelo SD fornece uma descrição intencional de um processo em termos de uma rede de relacionamentos de dependência entre atores relevantes, i.e. as relações de dependências externas entre os atores da organização. Esse modelo possui um conjunto de nós e ligações, em que cada nó representa um ator, e cada ligação entre dois atores indica que um ator depende de outro ator para algo. Os principais construtores do modelo SD na linguagem i* 1.0 são:

- **Atores**: especializados como *agent*, *role* e *position*;
- **Relacionamentos de especialização**: dos tipos *IsA*, *IsPartOf*, *Covers*, *Plays* e *Occupies*;
- **Relacionamento intencional**: representado por um *dependum* entre dois atores, o *depender* e o *dependee*. O *depender* é o ator que depende de outro ator (*dependeee*) para que o acordo seja realizado. O *dependum* pode ser utilizado para representar quatro tipos de dependências: *tasks*, *resources*, *goals* e *softgoals*.

Já o modelo SR descreve um processo em termos de elementos do processo e das razões que motivam a existência deles. As fronteiras de atores representam uma fronteira onde são explicitados os elementos intencionais desejados por um ator, como *tasks*, *resources*, *goals* e *softgoals*. Há também os relacionamentos *means-end*, *decomposition* e *contribution* (HORKOFF *et al.*, 2008).

2.2 ERROS FREQUENTES NA LINGUAGEM i*

Como em qualquer técnica de modelagem, é importante garantir que os modelos gerados sejam livres de erros. Modelos com erros podem prejudicar o entendimento do leitor impedindo a compreensão do que está sendo modelado. Para garantir que modelos não contenham erros, foram revisadas fontes da literatura que tratam de erros em modelos i* e, assim, criou-se um catálogo que agrupa em três categorias os 15 principais erros com uso da notação i* (SOUZA *et al.*, 2013):

1 **Atores e relacionamentos entre atores**

   a. Atores sem ligação, ou com ligação incorreta;
   b. Atores dentro da fronteira de outros atores;
   c. Uso de nomes inadequados em atores;
   d. Ligar atores com ligação de dependência sem usar um *dependum*.

2 **Dependências**

a. Uso de outras ligações em vez de ligação de dependência;
b. Uso inadequado de *dependums* (*softgoals* como metas, tarefas como *softgoals*, etc.);
c. Formação de ciclos entre *dependums* de atores diferentes.

3 **Elementos internos e relacionamentos entre elementos internos**

a. Deixar elementos sem ligação;
b. Ligação em situação irregular (dependência dentro da fronteira do ator);
c. Elementos do modelo SR fora de fronteira do ator correspondente;
d. Decompor metas em sub-metas ou sub-tarefas;
e. Decompor *softgoals* em *sub-softgoals* ou sub-tarefas;
f. Contribuição para metas, tarefas ou recursos;
g. *Means-Ends*, no qual uma meta é um meio;
h. Ligação direta entre elementos internos de dois atores diferentes.

2.3 ONTOLOGIA ONTOISTAR+ E FERRAMENTA TAGOOn+

A ontologia *OntoiStar+* (NAJERA *et al.*, 2013a) foi desenvolvida visando a integração das variantes da linguagem i*, como a *Tropos* e a *Service-Oriented i**, para propiciar o entendimento comum dessas variantes em relação à linguagem i*. Para cada variante foi criada uma ontologia na linguagem OWL (HITZLER *et al.*, 2012) com as definições sintática e semântica associadas às suas terminologias. A *OntoiStar+* unifica os construtores da linguagem i* e de suas variantes com a importação das ontologias citadas. Portanto, a ontologia *OntoiStar+* cumpre o objetivo de representar e integrar as variantes da linguagem i*, bem como de possibilitar a criação de modelos da linguagem, independentemente da variante na qual foram gerados (NAJERA *et al.*, 2013a).

A ferramenta *TAGOOn+* (NAJERA *et al.*, 2013b) permite realizar a transformação automática de modelos de requisitos representados em *iStarML* em uma instância da ontologia *OntoiStar+*, bem como a transformação automática de modelos expressos nas variantes mapeadas na *Ontoistar+*: i*, *Tropos* e *Service-Oriented i**. O processo de transformação é realizado recebendo como entrada um modelo i* em formato *iStarML*, e então a *TAGOOn+* faz um mapeamento de cada *<tag>* do *iStarML* para o seu respectivo construtor mapeado em OWL, segundo a ontologia *OntoiStar+*. O arquivo na sintaxe iStarML é analisado durante a execução da *TAGOOn+* a fim de aplicar regras de mapeamento para transformar um modelo em *iStarML* para um modelo em OWL. A vantagem é que esse arquivo em OWL gerado pode ser importado para um editor de ontologias para modificar, consultar e realizar inferências sobre a base de conhecimento utilizando um *plugin* do tipo raciocinador.

Brazilian Journal of Development

2.4 METODOLOGIA DE DESENVOLVIMENTO DA ONTOLOGIA ONTOISTAR-NG

Esta seção descreve a extensão da ontologia *OntoiStar+* chamada *OntoiStar-NG*. Com base nas regras de utilização dos construtores da linguagem i*, foram elaborados e incluídos axiomas e restrições na ontologia *OntoiStar+* visando não permitir que os erros catalogados pudessem ser modelados, gerando assim modelos mais consistentes. As restrições OWL foram inseridas na ontologia para definir limites para indivíduos que pertencem a uma classe.

Em cada subseção a seguir serão descritos o erro tratado com a respectiva sintaxe OWL que o inibe. A modelagem foi realizada na linguagem OWL 2, com a ferramenta *Protégé* 4.3 (MUSEN, 2015), integrada ao raciocinador *Pellet* 2.3.4 (SIRIN *et al.*, 2007) para reproduzir um modelo ontológico com o erro citado. Todos os exemplos seguem o catálogo de erros frequentes (SOUZA *et al.*, 2013) e a modelagem do cenário *Media Shop* (CASTRO; KOLP; MYLOPOULOS, 2002). O cenário de validação *Media Shop* inclui uma loja de vendas de diferentes tipos de itens (livros, jornais, revistas, etc), cujos clientes podem utilizar um catálogo periodicamente atualizado. Este cenário foi escolhido para validação por ser muito utilizado em trabalhos que tratam modelos i*.

Na representação do solução ontológica para tratamento de erros do referido catálogo serão usadas tabelas, onde a primeira coluna define a propriedade sobre a qual se aplica a restrição OWL descrita na segunda coluna[1]. Além disso, a representação _*-*ref* é utilizada para indicar as situações em que se deseja especificar a fonte (*_source-ref*) e o destino (*_target-ref*) de um relacionamento.

2.4.1 Tratamento Do Erro 1a

O erro 1a no catálogo de erros determina que não deve existir no modelo um ator sem uma ligação de dependência ou *ActorRelationship*. Além de não poder existir um ator sem ligações, os relacionamentos entre atores devem seguir as seguintes regras: *IsA* (relacionamento que só pode ocorrer entre atores do mesmo tipo); *Plays* (relacionamento apenas entre um agente e um papel); *Covers* (relacionamento apenas entre uma posição e um papel); *Occupies* (relacionamento apenas entre uma posição e um agente); e *INS* (relacionamento apenas entre dois agentes).

Para garantir que essas regras sejam modeladas de forma correta, foram inseridas na ontologia *OntoiStar-NG* as restrições descritas no Quadro 1.

[1] Por razões de espaço, são apresentadas apenas a descrição do erro e a regra incluída na *OntoiStar-NG*, porém todas as regras foram testadas na ferramenta *Protégé* similarmente às regras inseridas para mitigar o erro 1(a).

Brazilian Journal of Development

Quadro 1 - Restrições para garantir a existência de ligações corretas entre atores.

has_Actor_IsALink_*-ref	exactly 1 *Actor*
has_Actor_PlaysLink_*-ref	exactly 1 *Role*; exactly 0 *Agent*; exactly 0 *Position*
has_Actor_CoversLink_*-ref	exactly 0 *Role*; exactly 1 *Agent*; exactly 0 *Position*
has_Actor_OccupiesLink_*-ref	exactly 0 *Role*; exactly 1 *Agent*; exactly 0 *Position*
has_Actor_InstanceOfLink_*-ref	exactly 1 *Actor*

Fonte: elaborado pelos próprios autores.

2.4.2 Tratamento Do Erro 1b

Conforme boas práticas de utilização da linguagem i*, descritas em Malta *et al.* (2011) e Cares *et al.* (2011), não deve haver um ator dentro da fronteira de outro ator. Na ontologia *OntoiStar+* original, existe a classe *ActorBoundary*, que representa a fronteira de um ator, que precisou ser redefinida para que apresente *has_Actor_Boundary* e/ou *hasActor_Boundary_Elements* com exatamente 0 atores (ou seja, *owl:ObjectExactCardinality* igual a 0). Assim, o ator de um *ActorBoundary* será o único dentro do relacionamento, permitindo relacionamentos apenas com elementos da classe *InternalElement*.

Após essa redefinição ser inserida na *OntoiStar-NG*, não é mais possível ligar mais de um ator dentro da fronteira. Por exemplo, a ferramenta *Protégé* acusa uma inconsistência ao se tentar reproduzir o erro no qual os atores *Media* e *Bank_Cpy* deveriam ligar-se via relacionamento *has_Actor_Boundary* no cenário *Media Shop*.

2.4.3 Tratamento Dos Erros 1d E 2c

Não se deve usar um *link* de dependência entre dois atores sem um *Dependum*. Ou seja, um ator não pode estar diretamente ligado a outro ator. Para garantir que os erros 1d e 2c do catálogo de erros não ocorram, foram inseridas as seguintes redefinições nas classes *DependeeLink* e *DependerLink*, descritas no Quadro 2.

Quadro 2 - Restrições para que um *Depender* ou *Dependeee* não se conectem a outro *Actor*.

has_Dependency_DependerLink_*-ref	exactly 0 *Actor*
has_Dependency_DependeeLink_*-ref	

Fonte: elaborado pelos próprios autores.

Depois de inseridas estas restrições não é mais possível que um *Dependee* e um *Depender* conectem-se diretamente a outro ator, pois a restrição de cardinalidade especifica o número exato de relacionamentos possíveis entre os objetos especificados, ou seja, nenhum relacionamento com objetos da classe *Actor* deve existir com objetos das propriedades *has_Dependency_DependerLink* e *has_Dependency_DependeeLink*.

Brazilian Journal of Development

2.4.4 Tratamento Do Erro 2a

Deve-se usar *DependencyRelationships* somente para expressar dependência entre um membro da classe *Actor* e um *InternalElement*; não usar nenhum dos relacionamentos do tipo *InternalElementRelationShip* para representar dependências. Para garantir que esse erro não ocorra foram acrescentadas à classe *Actor* as restrições descritas no Quadro 3.

Quadro 3 - Restrições para que não haja relacionamentos do tipo *InternalElementRelationShip* para representar dependências entre *Actor* e *InternalElement*.

has_Dependency_DependerLink_source-ref	exactly 0 *InternalElement*
has_InternalElement_MeansEndLink_*-ref	
has_InternalElement_AndDecompositionLink_*-ref	
has_InternalElement_ContributionLink_*-ref	
has_InternalElement_DecompositionLink_*-ref	
has_InternalElement_OrDecompositionLink_*-ref	

Fonte: elaborado pelos próprios autores.

Para evitar o erro 2a, foi utilizada uma restrição de cardinalidade, ou seja, os objetos das classes *InternalElementRelationship* devem ter nenhum relacionamento com os objetos da classe *InternalElement*. Depois de inseridas essas regras não é mais possível gerar nenhum relacionamento do tipo *InternalElementRelationship* para representar uma dependência entre um *Actor* e um *InternalElement*.

2.4.5 Tratamento Dos Erros 2b E 2c

A classe *ActorBoundary* não deve possuir nenhum relacionamento direto do tipo *contribution* ou *dependency* com um *InternalElement*. Assim, foi incluída para a classe *ActorBoundary* as redefinições de classe descritas no Quadro 4.

As regras apresentadas no Quadro 4 evitam a ocorrência de qualquer relacionamento do tipo *DependencyRelationship* ou *InternalElementRelationship* com a classe *ActorBoundary*. Para atingir este objetivo foi utilizada uma restrição de cardinalidade que especifica o número exato de relacionamentos possíveis entre os objetos especificados, ou seja, os objetos das classes *DependencyRelationship* ou *InternalElementRelationship* não devem ter nenhum relacionamento com os objetos da classe *DependableNode*.

Brazilian Journal of Development

Quadro 4 - Restrições para que não conexão entre *InternalElement* e *ActorBoundary*.

has_Dependency_DependerLink_*-ref	exactly 0 *DependableNode*
has_Dependency_DependeeLink_*-ref	
has_Dependency_DependumLink_*-ref	
has_InternalElement_MeansEndLink_*-ref	
has_InternalElement_AndDecompositionLink_*-ref	
has_InternalElement_ContributionLink_*-ref	

Fonte: elaborado pelos próprios autores.

As regras foram criadas usando *DependableNode* em vez de *InternalElement*, pois assim garante que não poderão existir esses relacionamentos com itens da classe *InternalElement*, nem com a classe *Actor* que, conforme o erro 1b apresentado, também é um relacionamento indevido.

2.4.6 Tratamento Do Erro 3b

Não podem ser usados *links* de dependência dentro da fronteira de um ator. Ou seja, não devem ser realizadas relações de dependência entre um *InternalElement* dentro da fronteira de um ator. Pode haver um *Dependum* com origem externa à fronteira do ator ligando-se a um *InternalElement* de dentro da fronteira, mas nunca uma relação de dependência entre dois *InternalElement* internos à fronteira.

Para garantir que o erro 3b do catálogo de erros não ocorra foi acrescentada a redefinição na classe *ActorBoundary* da ontologia, a qual aplica uma restrição de cardinalidade igual a 0 a objetos da classe *InternalElement* via propriedade *has_Dependency_DependerLink_source-ref*.

Essa restrição especifica o número exato de relacionamentos possíveis entre os objetos especificados, ou seja, os objetos dos relacionamentos *has_Dependency_DependerLink_source-ref* devem possuir nenhum relacionamento (igual a 0) com os objetos da classe *InternalElement*.

2.4.7 Tratamento Dos Erros 3d E 3e

Uma meta não pode ser ligada diretamente a outra meta; esta deve ser decomposta utilizando uma ligação do tipo *MeansEndLink*, e estas ligações só podem ligar tarefas a metas. Essa definição segue o que determina o guia *iStarQuickGuide* versão 1.0. No Quadro 5 seguem as redefinições de classes para prevenir que os erros 3d e 3e ocorram.

Brazilian Journal of Development

Quadro 5 - Restrições para que uma meta possa ser decomposta apenas em tarefas e por meio de ligações do tipo *MeansEndLink*.

has_InternalElement_MeansEndLink_*-ref	exactly 0 *Goal*
	exactly 0 *Resource*
	exactly 0 *SoftGoal*
	exactly 0 *Plan*
	exactly 0 *Service*
	exactly 0 *Process*

Fonte: elaborado pelos próprios autores.

Depois de inseridas as restrições de cardinalidade, não é mais possível que uma meta possa ser ligada diretamente a outra meta. Estas restrições garantem uma limitação de cardinalidade exata sobre os relacionamentos possíveis entre os objetos especificados, ou seja, os objetos da classe *InternalElement_MeansEndLink* devem ter nenhum relacionamento com os objetos das classes *Goal*, *Resource*, *Softgoal*, *Plan*, *Service* ou *Process.*

2.4.8 Tratamento Do Erro 3f

Um *ContributionLink* só deve ser utilizado para conectar um *IntentionalElement* a uma *Softgoal*. Para garantir que esse erro não ocorra foi inserido na classe *ContributionLink* as restrições descritas no Quadro 6.

Quadro 6 - Restrições para que um *ContributionLink* conecte um *InternalElement* apenas a objetos da classe *SoftGoal*.

has_InternalElement_ContributionLink_*-ref	exactly 0 *Goal*
	exactly 0 *Plan*
	exactly 0 *Process*
	exactly 0 *Resource*
	exactly 0 *Service*
	exactly 0 *Task*

Fonte: elaborado pelos próprios autores.

Depois de inseridas essas restrições, não é possível utilizar um ContributionLink para conectar um *IntentionalElement* que não seja uma *Softgoal*. As restrições de cardinalidade inseridas garantem que não haja relacionamentos entre objetos das classes *Goal*, *Plan*, *Process*, *Resource*, *Service* e *Task* com elementos de *has_IntentionalElement_ContributionLink*.

2.4.9 Tratamento Do Erro 3g

Os *links* do tipo *MeansEnd* só podem ser utilizados para interligar *Task* a *Goal*. Para garantir que o erro 3g não ocorra e esse relacionamento seja utilizado de forma correta, foram inseridas restrições à classe *MeansEndLink* na ontologia *OntoiStar-NG*, tais como apresentadas no Quadro 7. Depois de inseridas essas redefinições de classe, não é possível utilizar um *MeansEndLink* para conectar um *InternalElement* que não seja uma *Task* a uma *Goal*.

Brazilian Journal of Development

Quadro 7 - Restrições para garantir que um *MeansEndLink* conecte apenas *Tasks* a *Goals*.

has_InternalElement_MeansEndLink_*-ref	exactly 0 *Resource*
	exactly 0 *SoftGoal*
	exactly 0 *Plan*
	exactly 0 *Service*
	exactly 0 *Process*

Fonte: elaborado pelos próprios autores.

Por fim, o Quadro 8 resume as restrições incorporadas à ontologia *OntoiStar-NG* em uma representação em lógica de primeira ordem, em que:

- *Ax* representa uma instância da classe *Actor*;
- um *InternalElement* é representado como *IE* ou (*T* ∨ *R* ∨ *G* ∨ *S* ∨ *PL* ∨ *SE* ∨ *PR*), com a correspondência *T* (*Task*), *R* (*Resource*), *G* (*Goal*), *S* (*Softgoal*), *PL* (*Plan*), *SE* (Service) e *PR* (*Process*).

2.5 DESENVOLVIMENTO DA FERRAMENTA TAGOOn-NG

As soluções desenvolvidas nesta pesquisa tiveram como principal objetivo possibilitar a geração de modelos de requisitos utilizando a linguagem i* livres de erros de má interpretação de seus construtores. Uma das etapas desta pesquisa foi estender a ferramenta *TAGOOn+* para também contribuir com a interoperabilidade entre modelos gerados com as variantes da linguagem i*.

Quadro 8 - Representação axiomática dos erros tratados pela ontologia *OntoiStar-NG*.

Erro	**Representação axiomática**
1a	A1 ^ (*has_Actor_IsALink** ^ (A2 ^ ¬(*Agent* ∨ Role ∨ *Position*)) A1 ^ (*has_Actor_PlaysLink** ^ (*Role* ^ ¬(*Agent* ∨ A2 ∨ *Position*)) A1 ^ (*has_Actor_CoversLink** ^ (*Role* ^ ¬(*Agent* ∨ A2 ∨ *Position*)) A1 ^ (*has_Actor_OccupiesLink** ^ (*Position* ^ ¬(*Agent* ∨ A2 ∨ *Role*)) A1 ^ (*has_Actor_InstanceOfLink** ^ (*Agent* ^ ¬(*Position* ∨ A2 ∨ *Role*))
1b	A1 ^ (*has_ActorBoundary* ∨ *has_Actor_boundary_elements* ^ (*T* ∨ *R* ∨ *G* ∨ *S* ∨ *PL* ∨ *SE* ∨ *PR* ^ ¬A2)
1d 2c	(*DependeeLink* ∨ *DependerLink*) ^ *has_Dependency_DependeeLink** ^ (¬A1)
2a	A1 ^ (*DependencyRelationship* ^ (¬*InternalElementRelationship*) ^ (*T* ∨ *R* ∨ *G* ∨ *S* ∨ *PL* ∨ *SE* ∨ *PR* ^ ¬A2)
2b 2c	A1 ^ (*has_ActorBoundary* ^ (¬(*DependencyRelationship* ∨ *InternalElementRelationship*)) ^ (*T* ∨ *R* ∨ *G* ∨ *S* ∨ *PL* ∨ *SE* ∨ *PR* ^ A2)
3b	*ActorBoundary* ^ (¬(*has_Dependency_DependerLink_source-ref*) ^ (*T* ∨ *R* ∨ *G* ∨ *S* ∨ *PL* ∨ *SE* ∨ *PR*)
3d 3e	A1 ^ (*has_InternalElement_MeansEndLink**) ^ (*G* ∨ ¬(*T* ∨ *R* ∨ *G* ∨ *S* ∨ *PL* ∨ *SE* ∨ *PR* ^ A2)
3f	*ContributionLink* ^ *has_InternalElement_ContributionLink** ^ (*S* ^ ¬(*T* ∨ *R* ∨ *G* ∨ *PL* ∨ *SE* ∨ *PR*)
3g	*MeansEndLink* ^ *has_InternalElement_MeansEndLink** ^ (*G* ^ ¬(*T* ∨ *R* ∨ *S* ∨ *PL* ∨ *SE* ∨ *PR*)

Fonte: elaborado pelos próprios autores.

Para iniciar esta etapa, foi realizado um estudo da referida ferramenta para posteriormente possibilitar a inserção das regras de validação em seu código, garantindo assim que os modelos gerados utilizando a ferramenta estejam livres dos erros que foram validados usando as restriçõeos OWL da *OntoiStar-NG*. O objetivo dessas alterações é permitir que a *TAGOOn-NG* seja capaz de

validar os arquivos em *iStarML* para que só sejam gerados arquivos OWL correspondentes, caso o modelo utilize os construtores da linguagem i* de forma correta. Caso contrário, a ferramenta deverá apresentar uma mensagem de erro solicitando correção antes de gerar novamente o modelo.

Inicialmente, foi realizado um estudo detalhado de cada método para entender sua funcionalidade e descobrir quais os métodos que necessitariam de alterações para atender as restrições da *OntoiStar-NG*. Após o entendimento das classes e métodos da *TAGOOn+* original, constatou-se que as alterações necessárias deveriam ser feitas nos métodos da classe *TransformationModule*, que é a responsável por transformar cada *tag* da linguagem *iStarML* em seu correspondente em OWL. Analisando esta classe, foram realizadas alterações nos cinco métodos a seguir, produzindo a ferramenta *TAGOOn-NG*, implementada com JDK 1.8.0_71:

- *LocateAndInstanceDependency*: foi acrescentada uma condição de teste para tratar o erro 3b, ou seja, para verificar se não existe dependência entre dois atores. Caso exista algum relacionamento de dependência entre dois atores no modelo em *iStarML* é gerada uma mensagem de erro para o usuário, informando que este tipo de relacionamento só é permitido entre elementos do tipo *Goal*, *Softgoal*, *Task*, *Plan* ou *Resource*.
- *LocateAndInstanceIElements*: para garantir que não ocorra o erro 1b, ou seja não deve existir um ator dentro da fronteira de outro ator, foi acrescentada uma condição de teste para verificar se o *ActorBoundary* é um elemento dos tipos *Goal*, *Softgoal*, *Task*, *Plan*, *Resource* ou *Process*. Caso contrário, é gerada uma mensagem de erro para notificar o usuário que aquele tipo não é permitido.
- *LocateAndInstanceDependums*: acrescentou-se uma condição de teste para garantir que não ocorram os erros 1d, 2a, 2b e 2c, verificando se não existe dependência direta entre dois atores, e se o relacionamento *Dependum* só permite a ligação entre *InternalElements*. Caso contrário, é gerada uma mensagem indicando o erro para que o usuário o corrija.
- *LocateAndInstanceBTWActorActorLink*: uma condição de teste foi introduzida para verificar se os relacionamentos entre os tipos de atores estão corretos. Garantindo que não aconteça o erro 1a quanto aos tipos de relacionamentos entre atores. Caso exista algum relacionamento errado ou algum ator sem ligação, será gerada uma mensagem de erro para que o usuário corrija o relacionamento em questão.
- *LocateAndInstanceBTWIelemenLink*: uma nova condição de teste foi adicionada para verificar se os relacionamentos do tipo *Decomposition*}, *Contribution* e *MeansEndLink* estão corretos, garantindo que não ocorram os erros 1d, 2c, 3f e 3g. Caso exista algum relacionamento incorreto, é gerada uma mensagem de erro para que o usuário o corrija.

Brazilian Journal of Development

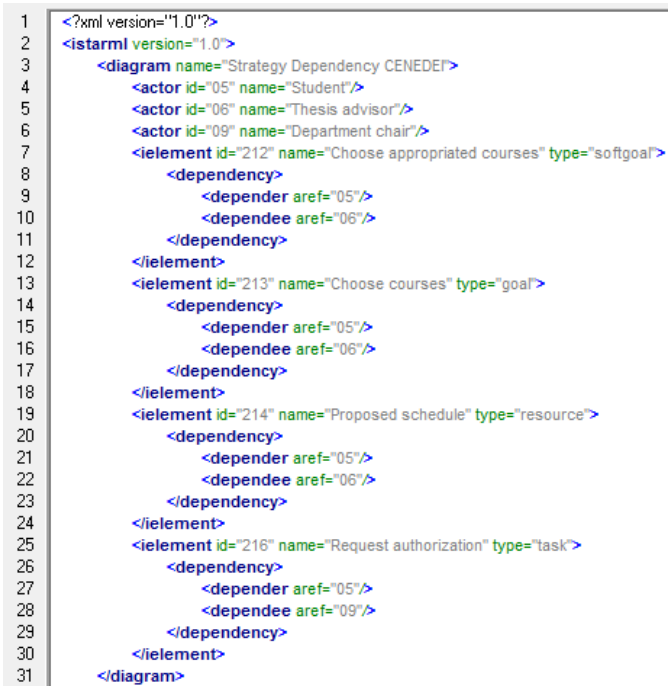
3 RESULTADOS E DISCUSSÃO

Após a inserção de cada condição nos métodos supracitados, foram realizados testes para verificar se a alteração iria garantir que não fossem gerados modelos OWL com os erros relatados na seção de tratamento de erros. Para realizar tais testes, foram criados arquivos *iStarML* com os erros listados no catálogo de erros frequentes, e verificando em seguida se a *TAGOOn-NG* recusaria o arquivo com erro e apresentaria a mensagem de erro para que o usuário o corrigisse.

Para ilustrar o funcionamento da *TAGOOn-NG*, a Figura 1, à esquerda, exibe um exemplo de arquivo *iStarML* sem erros, cujo conteúdo remete a conceitos modelados do cenário de validação de universidade (NAJERA, 2011). Nesse arquivo estão modelados três atores (estudante, orientador e chefe de departamento), três relacionamentos entre estudante e orientador (linhas 7, 13 e 19), e um relacionamento entre estudante e chefe de departamento (linha 25).

Já na Figura 1, à direita, tem-se o arquivo anterior escrito em *iStarML* adicionado da definição de fronteira com um serviço (linha 6) e outro ator (linha 7). Essa descrição remete ao erro 1b, no qual inclui-se um ator dentro da fronteira de outro ator. Ao submeter esse arquivo com o erro 1b à ferramenta *TAGOOn-NG*, ela não transforma o arquivo em uma instância da *OntoiStar-NG*. A Figura 2 descreve essa situação na qual retorna uma mensagem ao usuário informando o que está errado no arquivo de entrada para que possa ser corrigido e submetido novamente à ferramenta.

Figura 1 – À esquerda, um arquivo sem erros de sintaxe em *iStarML*, com conceitos do cenário de universidade; à direita, arquivo na sintaxe *iStarML* com o erro 1b, em que um ator encontra-se dentro da fronteira de outro ator.

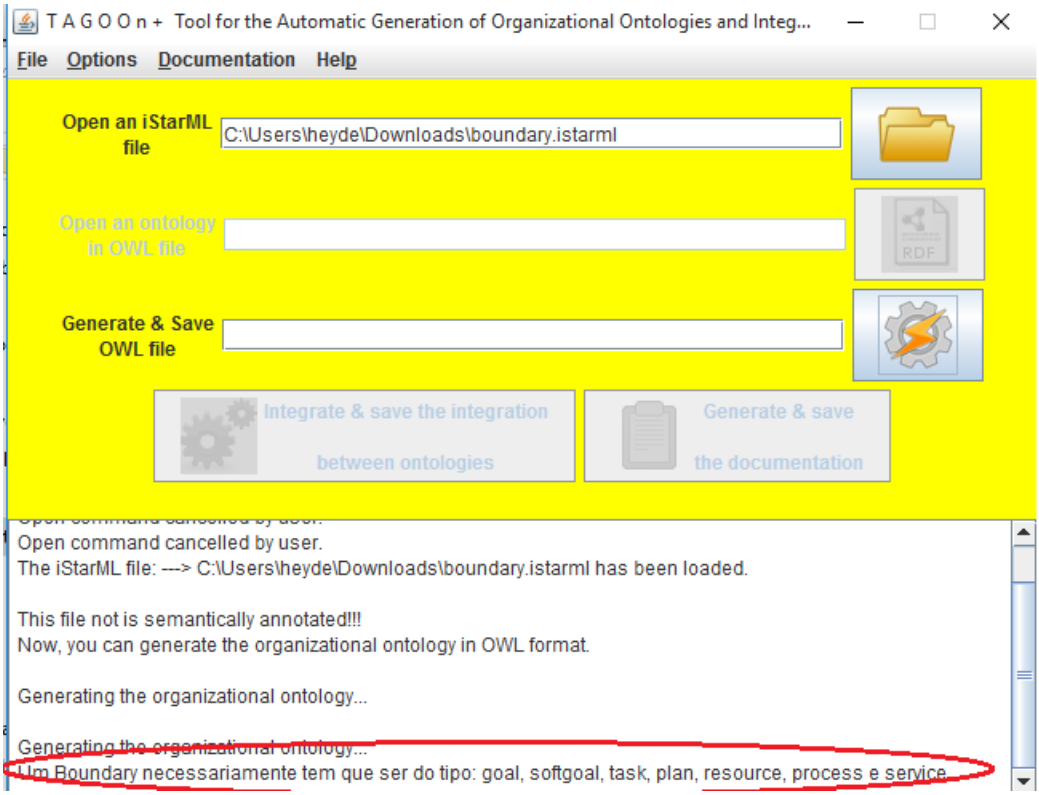


```
1  <?xml version="1.0"?>
2  <istarml version="1.0">
3      <diagram name="Strategy Dependency CENEDEI">
4          <actor id="05" name="Student"/>
5          <actor id="06" name="Thesis advisor"/>
6          <actor id="09" name="Department chair"/>
7          <ielement id="212" name="Choose appropriated courses" type="softgoal">
8              <dependency>
9                  <depender aref="05"/>
10                 <dependee aref="06"/>
11             </dependency>
12         </ielement>
13         <ielement id="213" name="Choose courses" type="goal">
14             <dependency>
15                 <depender aref="05"/>
16                 <dependee aref="06"/>
17             </dependency>
18         </ielement>
19         <ielement id="214" name="Proposed schedule" type="resource">
20             <dependency>
21                 <depender aref="05"/>
22                 <dependee aref="06"/>
23             </dependency>
24         </ielement>
25         <ielement id="216" name="Request authorization" type="task">
26             <dependency>
27                 <depender aref="05"/>
28                 <dependee aref="09"/>
29             </dependency>
30         </ielement>
31     </diagram>
```

```
1  <?xml version="1.0"?>
2  <istarml version="1.0">
3      <diagram name="Strategy Dependency CENEDEI">
4          <actor id="05" name="Student">
5              <boundary>
6                  <ielement id="104" name="Analyze courses" type="service"/>
7                  <ielement id="105" name="Authorize schedule" type="actor"/>
8              </boundary>
9          </actor>
10         <actor id="06" name="Thesis advisor"/>
11         <actor id="09" name="Department chair"/>
12         <ielement id="212" name="Department chair 1" type="softgoal">
13             <dependency>
14                 <depender aref="05"/>
15                 <dependee aref="06"/>
16             </dependency>
17         </ielement>
18         <ielement id="213" name="Choose courses" type="goal">
19             <dependency>
20                 <depender aref="05"/>
21                 <dependee aref="06"/>
22             </dependency>
23         </ielement>
24         <ielement id="214" name="Proposed schedule" type="resource">
25             <dependency>
26                 <depender aref="05"/>
27                 <dependee aref="06"/>
28             </dependency>
29         </ielement>
30         <ielement id="216" name="Request authorization" type="task">
31             <dependency>
32                 <depender aref="05"/>
33                 <dependee aref="09"/>
```

Brazilian Journal of Development

Figura 2 – Interface da ferramenta *TAGOOn-NG* exibindo mensagem, em destaque, referente ao erro 1b, o que evita produzir um arquivo OWL correspondente ao arquivo *iStarML* de entrada com esse erro.



Todos os 11 erros tratados foram testados de modo que um modelo em *iStarML* com o erro em questão era gerado e, utilizando a *TAGOOn-NG*, tentava-se converter esse modelo em uma instância da ontologia *OntoiStar-NG*. Quando a ferramenta encontra um dos erros tratados nesta pesquisa, ela informa imediatamente o usuário.

Em particular, no caso do erro 1a, ele foi tratado na ferramenta *TAGOOn-NG*, ou seja, não é possível gerar modelos ontológicos com este erro. Entretanto, as restrições OWL da *OntoiStar-NG* e a respectiva implementação na *TAGOOn-NG* garantem apenas que os relacionamentos estarão corretos, mas não garante que no modelo não existirá um ator sem ligação ou com ligação incorreta. Sendo assim, o erro 1a é coberto apenas parcialmente. Apesar de mesmo com a inserção das regras não conseguir validar o erro 1a, o impacto é minimizado já que este é o erro mais fácil de ser verificado visualmente no modelo, basta verificar se existe algum construtor do tipo *Actor* sem nenhuma ligação com outro construtor.

A grande vantagem da *TAGOOn-NG* é a possibilidade de validar modelos i* previamente desenvolvidos em *iStarML*, transformando estes em uma instância válida da *OntoiStar-NG*. A partir dessa instância, é possível utilizar o *Protégé* e o raciocinador *Pellet* para continuar adicionando e validando extensões ao modelo.

Brazilian Journal of Development

Embora a validação tenha sido realizada em apenas dois domínios diferentes, um domínio já seria suficiente para demonstrar a efetividade da solução *OntoiStar-NG* e *TAGOOn-NG*. Isto se deve ao fato de que as restrições OWL inseridasna ontologia servem para garantir a correta utilização dos construtores da linguagem, enão para garantir que a modelagem do domínio escolhido está correta. Ou seja, com esta solução é garantido que o modelo construído utilizando a linguagem i* não possua 11 dos 15 erros frequentes. Porém, mesmo assim a modelagem dos requisitos pode não estar correta. Desta forma, o domínio escolhido não influência nos resultados.

4 CONCLUSÕES E TRABALHOS FUTUROS

Dados problemas comuns de qualidade de modelos de requisitos escritos na linguagem i*, este artigo propôs a ontologia *OntoiStar-NG*, de modo a minimizar a quantidade de erros frequentes em modelos de requisitos dessa natureza e, assim, melhorar a sua qualidade. A *OntoiStar-NG* verifica a consistência dos modelos de requisitos escritos em i* à medida em que são criados através da utilização das restrições OWL inseridas na mesma, não permitindo a criação de modelos com erros frequentes, além de suportar a interoperabilidade entre variantes da linguagem i*.

A *OntoiStar-NG* foi integrada à ferramenta *TAGOOn-NG*, que permite transformar modelos i* escritos em *iStarML* em modelos baseados na estrutura e semântica dessa ontologia. A principal vantagem consiste na verificação automática de erros de modelos i* em *iStarML*.

Em relação à *OntoiStar+* original, foram adicionados 124 novos axiomas (25% axiomas a mais), que correspondem às restrições OWL para mitigação dos erros. Com a inserção dessas restrições, 11 dos 15 erros listados no catálogo de erros frequentes em modelos i* foram tratados, considerando o tratamento parcial dado ao erro 1a. Apenas 4 dos 15 não foram contemplados pela *OntoiStar-NG*: o uso de nomes inadequados em atores (1c); elementos sem ligações (3a); elementos do modelo SR fora de fronteira de ator correspondente (3c); e ligação direta entre elementos internos de dois atores diferentes (3h). Isto se deve ao fato desses erros serem mais subjetivos (por ex. uso de nome inadequado para ator) e/ou mais difíceis de tratar apenas com a expressividade dos construtores da linguagem OWL.

Como possibilidades de trabalhos futuros, pode-se realizar uma validação da *OntoiStar-NG* com especialistas em i*, um estudo sobre catálogos de erros para verificar se existem erros além dos já catalogados, bem como uma alternativa para tratar os erros que não puderam ser mitigados com a semântica da linguagem OWL. Para este último caso, pode-se adotar a linguagem de regras SWRL (*Semantic Web Rule Language*), que estende a capacidade da linguagem OWL na representação de conhecimento e na capacidade de inferência (DERMEVAL *et al.*, 2016).

Seria oportuno também realizar um estudo aprofundado da linguagem *iStar* 2.0, o que incluiria analisar as suas limitações à luz da expressividade de seus construtores e da interpretação desses construtores por desenvolvedores (DALPIAZ; FRANCH; HORKOFF, 2016). Dadas as modificações da *iStar* 2.0, é provável que alguns erros precisem ser revisados, como o erro 1a, já que a *iStar* 2.0 simplificou o relacionamento entre atores ao criar o relacionamento *Participates-in* e eliminar os relacionamentos *IsPartOf*, *Plays*, *Occupies* e *Covers*.

AGRADECIMENTOS

Os autores agradecem a todos aqueles que contribuíram com a realização deste trabalho.

REFERÊNCIAS

CARES, C.; FRANCH, X.; PERINI, A.; SUSI, A. Towards interoperability of i* models using iStarML. **Computer Standards & Interfaces**, Elsevier Science Ltd., Oxford, UK, v. 33, n. 1, p. 69–79, jan. 2011.

CASTRO, J.; KOLP, M.; MYLOPOULOS, J. Towards requirements-driven information systems engineering: The Tropos project. **Information Systems**, Elsevier Science Ltd., Oxford, UK, v. 27, n. 6, p. 365–389, set. 2002.

DALPIAZ, F.; FRANCH, X.; HORKOFF, J. iStar 2.0 Language Guide. 2016. ArXiv:1605.07767. Disponível em: <https://arxiv.org/abs/1605.07767>. Acesso em: 10 fev. 2020.

DERMEVAL, D.; VILELA, J.; BITTENCOURT, I.I.; CASTRO, J.; ISONTANI, S.; BRITO, P.; SILVA, A. Applications of ontologies in requirements engineering: A systematic review of the literature. **Requirements Engineering**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 21, n. 4, p. 405–437, nov. 2016.

GUIZZARDI, R. S. S.; FRANCH, X.; GUIZZARDI, G. Applying a foundational ontology to analyze means-end links in the i* framework. In: SIXTH INTERNATIONAL CONFERENCE ON RESEARCH CHALLENGES IN INFORMATION SCIENCE, Valencia, Spain, May 16-18 2012. [s.n.], 2012. p. 1–11.

HITZLER, P.; KRÖTZSCH, M; PARSIA, B.; PATEL-SCHNEIDER, P.F.; RUDOLPH, S. OWL 2 Web Ontology Language Primer (Second Edition), W3C Recommendation 11 December 2012. 2012. Disponível em: <http://www.w3.org/TR/owlprimer>. Acesso em: 10 fev. 2020.

HORKOFF, J.; ELAHI, G.; ABDULHADI, S.; YU, E. Reflective analysis of the syntax and semantics of the i* framework. In: ADVANCES IN CONCEPTUAL MODELING - CHALLENGES

AND OPPORTUNITIES, ER 2008 WORKSHOPS, Barcelona Spain, October 20-23, 2008. Proceedings. [s.n.], 2008. p. 249–260.

HORKOFF, J.; YU, E. Detecting judgment inconsistencies to encourage model iteration in interactive i* analysis. In: PROCEEDINGS OF THE 5TH INTERNATIONAL I* WORKSHOP 2011, Trento, Italy, August 28-29, 2011. [s.n.], 2011. p. 20–25.

MALTA, A.; SOARES, M; SANTOS, E; PAES, J.; ALENCAR, F.; CASTRO, J. iStarTool: Modeling requirements using the i* framework. In: PROCEEDINGS OF THE 5TH INTERNATIONAL I* WORKSHOP 2011, Trento, Italy, August 28-29, 2011. [s.n.], 2011. p. 163–165.

MUSEN, M. A. The Protégé project: A look back and a look forward. **AI Matters**, v. 1, n. 4, p. 4–12, 2015.

NAJERA, K.; MARTINEZ, A.; PERINI, A.; ESTRADA, H. An ontology-based methodology for integrating i* variants. In: PROCEEDINGS OF THE 6TH INTERNATIONAL I* WORKSHOP 2013, Valencia, Spain, June 17-18, 2013. [s.n.], 2013. p. 1–6.

NAJERA, K.; VASQUEZ, B; MARTINEZ, A.; PERINI, A.; ESTRADA, H.; MORANDINI, M. Tagoon+: Generation and integration of organizational ontologies. In: PROCEEDINGS OF THE 6TH INTERNATIONAL I* WORKSHOP 2013, Valencia, Spain, June 17-18, 2013. [s.n.], 2013. p. 125–127.

NAJERA, K. **An ontology-based approach for integrating i* variants**. 2011. 129f. Dissertação. National Center of Research and Technological Development, Cuernavaca, Morelos, Mexico, nov. 2011.

SIRIN, E.; PARSIA, B.; CUENCA GRAU, B.; KALYANPUR, A.; KATZ, Y. Pellet: A practical OWL-DL reasoner. **Web Semantics**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, v. 5, n. 2, p. 51–53, jun. 2007.